CLAIMS

What is claimed is:

1. A method for determining vectorization configurations in a computer processor architecture, the method comprising:

identifying a vectorizable loop in a computer program;

identifying a memory access pattern of data required for implementing said loop in said architecture;

computing a set of candidate configurations of resources required for vectorizing said data in said architecture, wherein said computing step comprises configuring a vector pointer register of said architecture in support of either of reorder-on-read use and reorder-on-write use of a vector element file of said architecture;

selecting one of said candidates in accordance with predefined selection criteria; and

implementing said selected vectorization configuration in said architecture.

2. A method according to claim 1 where any of said steps are implemented by a compiler.

3. A method according to claim 1 wherein said computing step comprises configuring any of said vector pointer registers in support of loading a data vector into a plurality of non-contiguous segments of said vector element file of said architecture.

4. A method according to claim 1 wherein said computing step comprises configuring any of said vector pointer registers in support of loading a data vector into said vector element file of said architecture in support of a plurality of operations where each operation has a different access pattern.

5. A method according to claim 4 and further comprising:

performing any of said steps for a plurality of vectorizable loops in the same computer program;

detecting a data reuse opportunity common to two or more of said loops; and modifying any of said candidate configuration in support of said data reuse opportunity.

6. A method according to claim 1 and further comprising eliminating any of said candidates in accordance with predefined elimination criteria.

7. A method according to claim 6 wherein said eliminating step comprises eliminating any of said candidates that requires loading a data vector into said vector element file in a manner that cannot be accommodated by said vector element file.

8. A method according to claim 1 wherein said selecting step comprises selecting one of said candidate configurations that uses fewest vector pointer registers among all of said candidates.

9. A method for determining vectorization configurations in a computer processor architecture for computations that feature arbitrary parametric access, the method comprising:

identifying a loop in a computer program that accesses data indirectly;

determining that indices of said loop fit within the range of a vector element file of said architecture, and, if so:

loading all loop data into said vector element file;

loading said indices into at least one vector pointer register of said architecture in support of reorder-on-read use of said vector element file; and

vectorizing said loop data.

10. A method according to claim 9 wherein said identifying step comprises identifying said loop as performing a plurality of computations that operate in parallel on a permutation of data.

11. A method according to claim 9 where any of said steps are implemented by a compiler.

12. A system for determining vectorization configurations in a computer processor architecture, the system comprising:

means for identifying a vectorizable loop in a computer program;

means for identifying a memory access pattern of data required for implementing said loop in said architecture;

means for computing a set of candidate configurations of resources required for vectorizing said data in said architecture, wherein said means for computing step is operative to configure a vector pointer register of said architecture in support of either of reorder-on-read use and reorder-on-write use of a vector element file of said architecture;

means for selecting one of said candidates in accordance with predefined selection criteria; and

means for implementing said selected vectorization configuration in said architecture.

13. A system according to claim 12 where any of said means are assembled with a compiler.

14. A system according to claim 12 wherein said means for computing is operative to configure any of said vector pointer registers in support of loading a data vector into a plurality of non-contiguous segments of said vector element file of said architecture.

15. A system according to claim 12 wherein said means for computing is operative to configure any of said vector pointer registers in support of loading a data vector into said vector element file of said architecture in support of a plurality of operations where each operation has a different access pattern.

16. A system according to claim 15 and further comprising:

means for performing any of said steps for a plurality of vectorizable loops in the same computer program;

means for detecting a data reuse opportunity common to two or more of said loops; and

means for modifying any of said candidate configuration in support of said data reuse opportunity.

17. A system according to claim 12 and further comprising means for eliminating any of said candidates in accordance with predefined elimination criteria.

18. A system according to claim 17 wherein said means for eliminating is operative to eliminate any of said candidates that requires loading a data vector into said vector element file in a manner that cannot be accommodated by said vector element file.

19. A system according to claim 12 wherein said means for selecting is operative to select one of said candidate configurations that uses fewest vector pointer registers among all of said candidates.

20. A system for determining vectorization configurations in a computer processor architecture for computations that feature arbitrary parametric access, the system comprising:

means for identifying a loop in a computer program that accesses data indirectly;

means for determining that indices of said loop fit within the range of a vector element file of said architecture, and, if so:

means for loading all loop data into said vector element file;

means for loading said indices into at least one vector pointer register of said architecture in support of reorder-on-read use of said vector element file; and

means for vectorizing said loop data.

21. A system according to claim 20 wherein said means for identifying is operative to identify said loop as performing a plurality of computations that operate in parallel on a permutation of data.

22. A system according to claim 20 where any of said means are assembled with a compiler.

23. A computer program embodied on a computer-readable medium, the computer program comprising:

a first code segment operative to identify a vectorizable loop in a computer program;

a second code segment operative to identify a memory access pattern of data required for implement said loop in said architecture;

a third code segment operative to compute a set of candidate configurations of resources required for vectorizing said data in said architecture and configure a vector pointer register of said architecture in support of either of reorder-on-read use and reorder-on-write use of a vector element file of said architecture;

a fourth code segment operative to select one of said candidates in accordance with predefined selection criteria; and

a fifth code segment operative to implement said selected vectorization configuration in said architecture.

24. A computer program according to claim 23 where any of said code segments are assembled with a compiler.

25. A computer program according to claim 23 wherein said third code segment is operative to configure any of said vector pointer registers in support of loading a data vector into a plurality of non-contiguous segments of said vector element file of said architecture.

26. A computer program according to claim 23 wherein said third code segment is operative to configure any of said vector pointer registers in support of loading a data vector into said vector element file of said architecture in support of a plurality of operations where each operation has a different access pattern.

27. A computer program according to claim 26 and further comprising:

a sixth code segment operative to perform any of said steps for a plurality of vectorizable loops in the same computer program;

a seventh code segment operative to detect a data reuse opportunity common to two or more of said loops; and

an eighth code segment operative to modify any of said candidate configuration in support of said data reuse opportunity.

28. A computer program according to claim 23 and further comprising a ninth code segment operative to eliminate any of said candidates in accordance with predefined elimination criteria.

29. A computer program according to claim 28 wherein said ninth code segment is operative to eliminate any of said candidates that requires loading a data vector into said vector element file in a manner that cannot be accommodated by said vector element file.

30. A computer program according to claim 23 wherein said fourth code segment is operative to select one of said candidate configurations that uses fewest vector pointer registers among all of said candidates.

31. A computer program embodied on a computer-readable medium, the computer program comprising:

a first code segment operative to identify a loop in a computer program that accesses data indirectly;

a second code segment operative to determin that indices of said loop fit within the range of a vector element file of said architecture, and, if so, load all loop data into said vector element file;

a third code segment operative to load said indices into at least one vector pointer register of said architecture in support of reorder-on-read use of said vector element file; and

a fourth code segment operative to vectorize said loop data.

32. A computer program according to claim 31 wherein said first code segment is operative to identify said loop as performing a plurality of computations that operate in parallel on a permutation of data.

33. A computer program according to claim 31 where any of said code segments are assembled with a compiler.